# AI505/AI801, Optimization – Exercise Sheet 05

2026-03-11

Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol $^*$ will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

## Exercise 1 $^+$ (7.1)

Direct methods are able to use only zero-order information, that is, only evaluations of $f$. How many evaluations are needed to approximate the derivative and the Hessian of an $n$-dimensional objective function using finite difference methods? Why do you think it is important to have zero-order methods?

## Exercise 2

Below you find an implementation of Nelder-Mead algorithm in Python. Analyze it and use it to solve a few Benchmark functions for unconstrained continuous optimization. You can use the Python implementations.

Plot the evolution of the simplex throughout the search (you can get help from AI assistants to code the plotting facilities).

Then, compare the results with the implementation of Nelder-Mead in the scipy library. You can use the COCO test suite to carry out this part or write yourself everything you need.

```python
import numpy as np

def nelder_mead(f, S, eps, max_iterations, alpha=1.0, beta=2.0, gamma=0.5):
    delta = float("inf")
    y_arr = np.array([f(x) for x in S])
    simplex_history = [S.copy()]
    iterations=0
    while delta > eps and iterations <= max_iterations:
        iterations+=1
        # Sort by objective values (lowest to highest)
        p = np.argsort(y_arr)
        S, y_arr = S[p], y_arr[p]
        xl, yl = S[0], y_arr[0]  # Lowest
        xh, yh = S[-1], y_arr[-1]  # Highest
        xs, ys = S[-2], y_arr[-2]  # Second-highest
        xm = np.mean(S[:-1], axis=0)  # Centroid

        # Reflection
```

```
        xr = xm + alpha * (xm - xh)
        yr = f(xr)

        if yr < yl:
            # Expansion
            xe = xm + beta * (xr - xm)
            ye = f(xe)
            S[-1], y_arr[-1] = (xe, ye) if ye < yr else (xr, yr)
        elif yr >= ys:
            if yr < yh:
                xh, yh = xr, yr
                S[-1], y_arr[-1] = xr, yr
            # Contraction
            xc = xm + gamma * (xh - xm)
            yc = f(xc)
            if yc > yh:
                # Shrink
                for i in range(1, len(S)):
                    S[i] = (S[i] + xl) / 2
                    y_arr[i] = f(S[i])
            else:
                S[-1], y_arr[-1] = xc, yc
        else:
            S[-1], y_arr[-1] = xr, yr

        simplex_history.append(S.copy())
        delta = np.std(y_arr, ddof=0)

    return S[np.argmin(y_arr)], simplex_history
```

## Exercise 3 *

The Nelder-Mead algorithm has three parameters $\alpha, \beta$, and $\gamma$. How would you approach the problem of tuning these parameters?

## Exercise 4 *

Consider the natural evolutionary strategy for an univariate function. Assume the univariate normal distribution as proposal distribution $p(x \mid \boldsymbol{\theta}) = N(x \mid \mu, \sigma^2)$.

- Derive the update rule for $\theta$
- If after a number of iterations the value of $\mu$ becomes equal to $x^*$, that is, the minimum of $f$, what will be the update rule for $\sigma^2$ and what will be the difficulty encountered by the algorithm?

## Exercise 5 * (8.4)

The maximum likelihood estimates are the parameter values that maximize the likelihood of sampling the points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$.

In the slides it was given the analytical solution for the mean and variance of a multivariate Gaussian distribution: $N(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma)$.

Derive here those results that are used in the cross-entropy method that uses multivariate normal distributions.

## Exercise 6 *

Implement Simulated Annealing. Set the initial temperature such that the initial acceptance ratio is 0.2 and the annealing plan to exponential with cooling rate $\gamma = 0.99$. Apply the algorithm to the Rosenbrock function and plot the value of the function and the temperature throughout the iterations.